Australian Government
Department of Defence
Defence Science and
Technology Organisation

Evaluation of Software Dependability at the Architecture Definition Stage

Kiril Uzunov and Thong Nguyen

Air Operations Division
Defence Science and Technology Organisation

DSTO-TR-2428

ABSTRACT

The problem we aim to solve is how to evaluate the dependability of software at the stage of architecture definition. Evidence, such as the process maturity, project environment and architecture documentation is already available and can be used for the evaluation. In order to create a holistic picture of the state of dependability, a Bayesian Network (BN) model is defined. The paper defines a quality framework which guides the model creation, identifies attributes characterising dependability and presents the topology of the model. The approach to the quantitative definition of the model is illustarted by examples. The model is aimed to help with conducting technical risk assessment of Airborne Mission Systems.

Published by

Air Operations Division
DSTO Defence Science and Technology Organisation
506 Lorimer St
Fishermans Bend, Victoria 3207 Australia

Telephone: (03) 9626 7000
Fax: (03) 9626 7999

AR-014-792
June 2010

APPROVED FOR PUBLIC RELEASE

Evaluation of Software Dependability at the Architecture Definition Stage

Executive Summary

The achievement of software quality goals needs to be considered during all stages of the software development process. Quality factors are effectively constrained by the architecture and therefore, they need to be evaluated as early as possible at the architecture definition phase. The quality factor we aim to evaluate is dependability of software.

The goals of software architecture evaluation at AOD / Airborne Mission Systems are to: (1) support software systems acquisition; (2) assess project health and predict project risks. The model described in this report aims to help the technical risk assessment by providing guidelines / metrics for evaluating the quality attributes associated with software dependability.

To harness the process of software dependability evaluation, we adopt the McCall quality model with some modification. The model represents a four level hierarchy of quality factors, Level1 and Level 2 quality attributes and metrics.

We aim to derive an estimate for each quality factor, where the quality factors defining dependability are *Reliability*, *Safety* and *Maintainability*. In order to achieve this, we build a "dependency graph" between the quality factors (the first level of the quality framework), level 1 quality attributes (testability, complexity, openness, robustness, modularity etc), and level 2 attributes characterising the engineering process, the organisation and the software architecture (process maturity, management practices, team experience, quality of requirements elicitation, architecture style, coupling, error propagation, capacity margin etc). Finally, checklists are formulated to reason about the quality attributes.

Bayesian Networks (BN) provide a mechanism to express the causal relationships between the elements of the quality framework. Bayesian Network is a graphical model that represents the dependency between the model's variables; these variables represent our quality attributes and quality factors. These relationships are causal, but our understanding of them is not complete, which is why it is possible to describe them probabilistically.

This report describes the different attributes forming the Bayesian model to be used for evaluation of software dependability: nodes (representing the quality factors and attributes) and Conditional Probabilities Tables (representing the probability given node taking each of its values). The end goal is to provide a tool supporting the technical risk assessment activities of mission system software in Defence acquisition projects.

Authors

Kiril Uzunov
Air Operations Division

Kiril Uzunov's research interests are in software dependability, sotware metrics and software engineering. His background is in software development (Electronic Design Automation software and real-time software), software testing and verification, and project management. Before joining DSTO in 2006, he spent 11 years with Freescale Semiconductors Inc. and Motorola Inc. as a Senior Software and Verification Engineer, Technical Lead, Project Lead and Program Manager. He has a Masters degree in Computer Systems Engineering.

Thong Nguyen
Air Operations Division

Dr Thong Nguyen holds BE (Honour I), BSc and PhD from Adelaide University and MBA (TechMgt) from Deakin University. He is currently the Research Program Manager for Airborne Mission Systems Branch, AOD, where he has been leading a team of 8 researchers and NICTA colleagues to develop expertise on Capability Integration. The current AMS research program employs different techniques such as AADL, Colour Petri Nets and Bayesian Networks to assess software quality at architecture level.

Contents

Acronyms

AA	Architecture Attributes
ADL	Architecture Description Language
ALMA	Architecture Tradeoff Analysis Method
AMS	Airborne Mission System
ARID	Active Review for Intermidiate Designs
ATAM	Architecture Tradeoff Analysys method
BN	Bayesian Network (also known as Belief Network)
CBAM	Cost-Benefit Analysis Method
CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-The Shelf
CPT	Conditional Probabilities Table
DMS	Diminishing Manufacturing Sources
DMSMS	Diminishing Manufacturing Sources and Material Shortages
DoSAM	Domain Specific Software Architecture Comparison Model
FAA	Federal Aviation Authority
FMEA	Failure Mode and Effect Analysis
FMECA	Failure Mode, Effect Analysis
GOTS	Government Off-The Shelf
GUI	Graphical User Interface
NFR	Non-Functional Requirements
OOD	Object Oriented Design
PCA	Principal Component Analysis
QUASAR	Quality Software Architecture Review
SAAM	Software Architecture Analysis Method
SACAM	Software Architecture Comparison Analysis Method
SDLC	Software Development Life Cycle
SIL	Safety integrity level(s)

1. Introduction

1.1 Software Architecture and Quality of Software

Software is the critical enabling technology for consumer electronics as well as for security and safety critical applications used in avionics, space, railway and transport, process control and medical systems. This trend finds its expression in monetary terms; for example the UK Defence Procurement Agency spends over £1Bn of its £7Bn annual budget on software, and the percentage spent on software will only grow [MoD 2006]. The increasing role software plays in modern day technology puts software quality in the limelight.

The term software quality is a generic term which needs a general consensus on meaning [Voas 2008]. It represents the notion of software being fit for a purpose, i.e. the things the user / customer is expecting from the product. These expected things are not functional requirements – implementing the functional requirements is well defined. It has been a number of years since it was pointed out that the non-functional requirements (NFR), also known as "ilities", are the ones which are the challenge and lead to the blow-up in cost and schedule. Functionality and quality attributes are orthogonal, because otherwise the choice of a function would define the level of a certain quality attribute [Bass et al. 2003].

To harness the process of software development toward achieving manageable quality goals, these goals had to be measured, and, as a result, quality measurement frameworks were proposed. The McCall quality model (illustrated in Figure 1) is based on three types of quality characteristics: Quality Factors, Quality Criteria and metrics [McCall 1994].

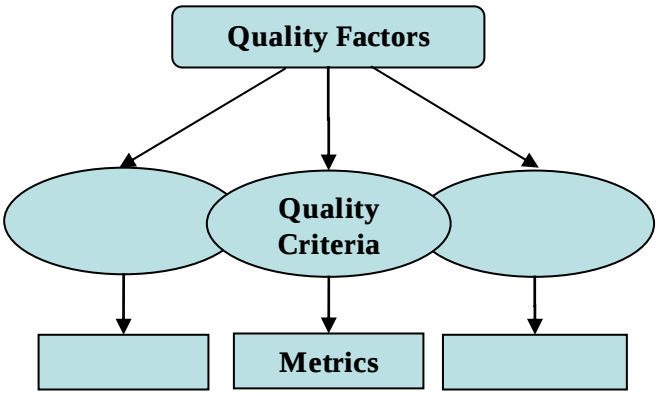

Figure 1: Quality measurement framework

A *quality factor* (also called quality characteristic in ISO / IEC 9126 [IEC 9126]) is a quality goal and represents a characteristic of the software that a customer would relate to the overall quality. Consequently, this characteristic reflects the external (user) point of view and would typically be given in the requirements. Some studies apply the term *quality attribute* [Florentz et al. 2006] for this characteristic, which introduces some confusion.

The second level of the software quality framework – *quality criteria*, represents / provides software product attributes related to the quality factors. These attributes reflect the internal (developer) point of view. The third level of the quality framework consists of metrics associated with the criteria. These metrics define whether the criteria exists and to what degree. They can be checklists or review / inspection guidelines which "grade" the quality criteria or quantitative measures of characteristics such as size, complexity etc.

The quality framework provides a basis for a disciplined approach to assess software quality. The logical steps to be undertaken within such a goal-oriented approach are:

- Identify quality factors we are interested in;
- Consider what criteria impact these quality factors;
- Consider the interdependencies between the factors (common criteria which may have positive impact on one factor while having negative impact on another factor);
- Provide measurements (metrics program) to assure the quality criteria is built into the software product; and
- Track, analyse and improve metrics collection.

The achievement of the software quality goals needs to be considered during all stages of the software development life cycle. We are interested in the early phases of the development – up to and including the software architecture definition phase which is part of the software design. Software design fits between the software requirements analysis and software construction and is considered a two step activity: architectural (or high level design) design and detailed design [IEEE / EIA 12207], [SWEBOK 2004]. Architectural design describes how software is decomposed and organised into components (the software architecture).

Software architecture is at the centre stage in modern software engineering as the platform for making major decisions which will have long term impact on all consequent artefacts, like mapping functionality to hardware and software, configuration of components, breakdown of interfaces etc. Although architecture by itself is not able to achieve quality, it provides the basis for achieving quality [Bass et al. 2003]. Quality factors are effectively constrained by the architecture and therefore, they need to be evaluated at the architecture level.

The goals of software architecture evaluation for Airborne Mission Systems (AMS) are to: (1) support software systems acquisition; (2) assess project health and predict project risks. The assumed relations between design solutions and quality requirements are not always correct [Bosch et al. 2001]. Detailed evaluation giving sufficient insight in the attributes of an architecture design is expensive, consuming considerable time and resources. The model we develop aims to help the technical risk assessment by providing guidelines / metrics for evaluating the quality attributes associated with dependability, as dependability is of highest priority in avionics systems.

1.2 Dependability

This report is concerned with the identification and evaluation of the dependability factors for Airborne Mission Systems. Dependability is the ability of a system to avoid service failures which are more frequent and more severe than expected [Avizienis et al. 2004]. Dependability

is a generic term which includes several quality factors: reliability, availability, maintainability, safety, and integrity. This report builds on [Uzunov et al. 2008], where a survey was undertaken to review the existing approaches to dependability assessment and improvement at various stages of the software development life cycle (SDLC).

Predicting dependability of such complex computer systems is difficult because of [Prasad 1998]:

- Problems of integration resulting from unexpected interaction between the large number of components;
- Multiple attributes characterising dependability;
- The scientific principles of measurement have not been applied with adequate rigour for decision making during the earlier stages of the development process.

The ultimate questions we seek to answer are: (1) What quality criteria, project and architecture characteristics contribute to the dependability factor and how do they interact; (2) How these characteristics can be captured / measured for a specific project; (3) How can we use these characteristics to predict the risk / quality of a specific project.

Because software architecture has critical impact on the quality factors (including dependability), we concentrate on the architecture level evaluation and issues stemming from the software development during requirements definition and architecture definition. In this report we discuss the quality factors, criteria and attributes impacting dependability, their interrelations and propose a model to be developed based on this. Chapter 2 lists the approaches used for software architecture evaluation, elaborates the characteristics of the Airborne Mission Systems influencing software dependability and presents the reasons for applying a Bayesian Network (BN) model to the evaluation of software dependability. Chapter 3 describes the Bayesian Network model – the semantics of the variables represented as nodes, the dependencies between nodes and describes how the Conditional Probabilities Tables are defined. Chapter 4 concludes.

2. Software Architecture Evaluation

2.1 Summary of Approaches

The architecture evaluation techniques can be grouped in: (1) architecture oriented techniques and (2) quality attribute focused techniques [Bosch et al. 2001].

The architecture oriented techniques are built around expert reviews following defined guidelines. Typically, the reviews are held after the architecture is defined. Most of these techniques are briefly described in [Bergner et al. 2005]. [Babar et al. 2004] proposes a framework for their comparison and assessment. Here are some examples of architecture oriented approaches:

- Software Architecture Analysis Method (SAAM) – Evaluation of software architectures is executed through scenarios, quality attributes and quality objectives.

- Architecture Tradeoff Analysis method (ATAM) – This method concentrates on identifying critical tradeoff points which will impact the architecture (including its quality atributes) and the risk involved. As a result, the advantages and disadvantages of each tradeoff are well understood.
- Cost-Benefit Analysis Method (CBAM) – builds up on ATAM by adding cost of architectural decisisons.
- Architecture Tradeoff Analysis Method (ALMA).
- Active Review for Intermidiate Designs (ARID).
- Software Architecture Comparison Analysis Method (SACAM).
- QUASAR [Firesmith 2006] - QUASAR system architecture assessment method is essentially an audit following the CMU SEI QUASAR methodology and is based upon quality cases. A quality case is a generalisation of a safety case consisting of claim, argument and evidence. The audit looks at whether the presented evidence proves that the architectural decisions and rationales lead to software architectures able to support the quality requirements.
- Quality Software Architecture Review – This method defines a process and guidelines for an architecture review by external reviewers.
- Domain Specific Software Architecture Comparison Model (DoSAM) [Bergner et al. 2005]. It is based on quality attributes and scenarios, but is tailored for the needs of a domain analysis. It introduces architectural services for the purpose of an abstratct description of the application domain.

The quality attribute focused techniques aim to come up with an estimate for each quality attribute or compare architectures against a given quality attribute. These techniques (which may be based on quantitative or qualitative assessment) can be grouped further into 3+1 methods: scenario-based, simulation, mathematical modeling / metrics and experience-based reasoning [Bosch 2001]. The brief descriptions of the process to be followed for each techniques is taken from [Lundberg et al. 1999].

- Scenario based evaluation – a profile for a specific quality attribute is created (a scenario profile is a set of typical scenarios, e.g. hazard scenarios for safety). Then the impact of the scenarions on the architecture is assessed. Based on this, a prediction can be made about the quality attribute. Such a technique for assessing the optimal maintainability is described in [Bosch et al. 2001].
- Simulation - the architecture is modelled (using architecture description languages or conventional languages) and the results from runing scenarios on the model are analysed in order to predict the quality attribute under investigation. An example of such an approach is [Gregoriades et al. 2005] where the system is described using the i* language and then scenarios are created based on interviews with the users and stakeholders. The scenarios (lists of linked tasks) are converted into executable form and a Bayesian Network model is plugged into the tool. The model takes as inputs the characteristics of each task and the environment and calculates the reliability for the given scenario.
- Mathematical modeling (including metrics) – the architecture is presented in terms of an appropriate mathematical model; the model output is calculated and interpreted in order to predict the quality attribute. This approach includes the collection of product

and / or process metrics which are used to make a prediction about a given quality attribute. Many studies are dedicated to the definition of various metrics and the validation of these metrics as predictors of specific qualities. [Shereshevsky et al. 2001] defines coupling and cohesion metrics characterising information flows in the architecture. These metrics are related to quality criteria like error propagation and change propagation. Design metrics reflecting connectivity of architecture components and the information in / out flows as well as component's internal structure are defined in [Stineburg et al. 2005]. The referenced report further studies the application of the design metrics for evaluating software reliability.

- Experience-based assessment is based on the experience of the designers who review the architecture looking for weaknesses against a given quality attribute.

Our approach to evaluating the dependability attributes of Airborne Mission Systems may be assigned formally to the group of methods based on mathematical modeling. We are working on a Bayesian Network model called AMS-BN (Air Mission System - Bayesian Network).

2.2 Applying Bayesian Networks to Dependability

The only means of direct evaluation of dependability is through operational testing of the software. In addition to the testing, other information is available for evaluation, which characterises the development organisation, the quality of the software engineering process applied during the Software Development Life Cycle and the engineering. Some of this information is publicly known, e.g. whether the organisation has been assessed against CMMI or ISO2000 or some other standard. Applicable standards can be identified from the requirements documentation. The project plans would provide information about the selected development process, the collection of metrics and would indicate whether the organisation really functions at the level of maturity at which it was assessed. Finally, most of the information concerning the engineering activities can be obtained as a result of an audit or request for information to the developer. In the case of the latter, the information needs to be identified and included as part of the contract negotiation. This information may include high-level design decisions as reflected in the software architecture documentation, evidence for and results from reviews, safety analysis, Failure and Effect Analysis and code inspections.

None of the evidence mentioned above can alone provide enough information about the dependability characteristics of software. Probably the most important thing for understanding the software reliability measurement is experience and judgement. Our conclusion is that the only way to build a holistic picture of the state of dependability is to develop a comprehensive framework, where all the quantitative and qualitative evidence described in this report (people, process and product) is captured. Such a framework needs to provide input, guidance and / or suggestions to the expert to make conclusions. The selected approach should be able to capture uncertainty, expert judgement and incomplete information. Such a framework can be used for assessment of dependability and consequently to help Technical Risk Assessment (TRA).

Various approaches are used to combine disparate evidence in order to make a valuation of the overall dependability of a system. The DATUM (Dependability Assessment of Safety Critical Systems through the Unification of Measurable Evidence) project in the UK (the Centre for Software Reliability, City University, London) investigated several formalisms used

to model uncertainty [Falla 1998]. The methods considered were Bayesian probability, Dempster-Shafer theory of belief functions, fuzzy sets and possibility theory. Although no single formalism for uncertainty was found perfect, Bayesian probability was chosen as the most mature and well developed formalism at the time [Fenton et al. 1998]. The obstacle for using Bayesian probability in cases of multiple evidence was the complex computations. This problem has been mostly overcome with the development of algorithms, solutions and tools in support of this formalism – the Bayesian Networks. The developments in network propagation algorithms make Bayesian inference computationally feasible for solving complex problems. Bayesian inference can also be used for "what if" analysis. Brief reviews and details for most of the available Bayesian Modelling tools can be found in [Anthony 2006] and [Murphy 2005]. It is interesting to note in this context the paper [Simon et al. 2006], where Bayesian Network implementation of the Dempster-Shafer theory is used to model the reliability uncertainty. [Ziv et al. 1997] also identified Bayesian Networks as a suitable technique for modelling uncertainty in software systems.

Bayesian Networks were also used by the FASGEP (Fault Analysis of the Software Generation Process) project to determine the fault propensity of software processes [Falla 1998]. The direction taken by the DATUM project has being followed in a series of projects within the RADAR (Risk Assessment and Decision Analysis Research) Group in Queen Mary University of London led by N.Fenton. [Wang et al. 2006] reported on the application of BN for project level estimation, where the accent is on the development and the test phases of the SDLC. [Perez-Minana et al. 2006] reported on the development of BN models for prediction of fault insertion and fault removal. The models were used as part of the software development process in Motorola, Toulouse. It is worth noting that the accuracy of the initial predictions of the generic models required a calibration based on measures from concrete projects. A procedure is suggested to calibrate the models and arrive at an improved BN. One approach was to vary the values associated with each node that are used as inputs to the intermediate nodes. The other approach (which produced better results) used linear regression and Principal Component Analysis to build the intermediate and the output nodes.

BN models have been used for scenario-based analysis and assessment of Non-Functional Requirements (including reliability) [Sutcliffe et al. 2002] and [Gregoriades et al. 2005]. The models are plugged into the System Requirements Analyser tool. Scenarios are designed and depending on the selected tasks, the technology attributes, the human attributes and the environment variables, the BN model provides evaluation of the reliability for the scenario.

Our research was inspired by the approach described in [Fenton et al. 2007] and also stems from [Gurp 2003], where a Baysean Net model for reasoning about software architecture attributes is developed. The model uses a hierarchy of quality factors, quality criteria and architecture attributes. The major differences to our model, considering our specifc focus on Airborne Mission Systems (AMS), is that a different set of quality criteria and architecture attributes have been identified. Our model also includes characteristics associated with the process and the project, since these have a major impact on the dependability of mission critical systems. Our work goes further in the quality framework by identifying metrics (in the form of guidelines) for qualitative assessement of some criteria, which are domain specific. Another analogue to our work can be seen in [Florentz et al. 2006], where the same fundamental approach to quality evaluation is observed, namely, a hierarchy of quality factors and quality criteria is identified, and relationships between architectural elements and

quality factors are investigated. The quoted technique uses a pragmatic and intuitive method of assigning quantitative values to the quality factors, but in comparison, our model offers more flexibility (e.g. Bayesian inference allows "what if" analysis).

We are interested in the relationships between the architecture and organisation attributes on one hand, and the dependability factors. These relationships are causal, but our understanding is not complete (or is uncertain), which is why we can describe them probabilistically. A Bayesian Network is a type of causal model which uses Bayesian probability. Bayesian Network is a graphical model - a directed acyclic graph, which captures probabilistic relationships between the model's variables; these variables represent our attributes and quality factors. The variables are represented by nodes, while the arcs (links) of the BN represent conditional interdependencies between the variables. Nodes without parents are called root nodes; they have a prior distribution associated with them. In our model most of the architecture and organisation attributes are represented by root nodes. Every node with parents is associated with a Conditional Probabilities Table (CPT) that represents the probabilities of that node taking each of its values, given the combinations of values of its parent nodes.

The general steps in constructing a BN are described in [Korb et al. 2004]. First the topology of the net is defined by identifying the variables – often starting from the root causes and adding new variables until the leaves of the net are reached. One node per variable is entered, the nodes are connected and their names and states are defined. The next step is the definition of dependencies between the nodes, i.e. filling the CPT. When the net is constructed, it can be applied to a specific case by entering the known values of the variables (i.e. evidence) into their corresponding nodes. A BN tool will do for us a probabilistic inference in order to find new beliefs (posterior probabilities) for all other variables.

Bayes' theorem is used as a basis for updating the information in a BN. The theorem calculates the probability of two (or more) dependent events A and B:

$$P(A,B) = P(B,A) = (p(B \mid A) * p(A)) / p(B),$$

where $p(A \mid B)$ is the probability of event B happening given A has already happened. Bayes' theorem may be too complex if an exact solution is required for a large number of events, however, for specific classes it can be efficiently solved. Many algorithms have also been developed for finding approximate solutions of the conditional probabilities in a Bayesian Network [Charniak 1991]. Once the CPT is filled in, the Bayesian Network ensures that the numbers will be consistent and the network will uniquely define a distribution.

Our beliefs in the software dependability factors are influenced by many attributes as illustrated in Figure 3. BNs accommodate for the combination of different variables: some of them precisely defined (e.g. defect containment for a specific phase) or some being evaluated qualitatively based on expert opinion or guidelines developed by experts. BNs provide a mechanism for combining the evidence from architecture and organisational attributes in order to calculate the probability that the dependability factors have a certain value. The network is updated as soon as new evidence is entered.

2.3 Characteristics of the AMS domain

An AMS is employed in a specific environment which imposes additional constraints or requirements in comparison to a general purpose computer system. Typically, the development is a collaborative effort of multiple partners and subcontractors with different culture as well as widely geographically dispersed teams. Well defined **standards** for documentation, interfaces and protocols are essential in this case. AMS are mission and safety critical, hence the maturity of the development **process** and **experience** of the organisation are a primary concern.

Air platforms (and their AMS) are in service for long periods of time in the range of tens of years. As a result, an enormous problem is that of unavailable electronic parts and the ageing of technologies. As pointed in [Sandborn 2008], an estimated 3% of the global pool of electronic components is discontinued each month. Electronic parts obsolescence – also known as Diminishing Manufacturing Sources (DMS) or Manufacturing Sources and Material Shortages (DMSMS) is an issue, since the majority of processor and memory chips in AMS are COTS units. In order to keep the cost of technical refreshes and upgrades under control, architecture should have characteristics like **maintainability**, **scalability** and **modifiability**.

The development cycle for AMS is longer than the current technology cycle and leads to technology ageing. This and the long lifetime of a platform, mean that the insatiable hunger for space, power and weight on an aircraft platform will result in requirements for new capability or substitution of existing blocks with new ones offering improved capability. This means also that spare processor capacity, bus bandwidth and PCB spaces have to be provided, together with an architecture which has to be **scalable** in order to incorporate the increased capacity. Another prerequisite for future upgrades is architecture **openness** (both for software and hardware), i.e. the architecture has to be based on publicly available widely accepted standards.

Changes in the underlying hardware go hand in hand with the necessity to port or rewrite the associated software. Thus upgrades and technology refreshes require appropriate **partitioning** of the software and mapping to hardware so that any changes will not cause unexpected changes in other parts of the system. Appropriate portioning at different levels can enable incremental integration which decreases the integration risks compared with the big-bang approach. The selection of an appropriate **architecture pattern** will help isolate changes in the hardware to a specific software block.

Airborne Mission Systems (as well as any new military systems) are characterised by growing **complexity** and size. The high complexity of the systems is made even more complex with the trend toward interoperability of systems which have not been designed to talk to each other. The reliance of operations on information processing and networks led to the requirement for all US DoD acquisition programs to address interoperability and integration [DoD DAG].

3. Airborne Mission Systems Bayesian Network Model

3.1 Topology of the Model

This section concentrates on the first step of constructing a BN – building the topology of a BN which represents how the dependability factors are interconnected with the architecture quality criteria. We will identify the major quality factors and architecture criteria (attributes) contributing to the dependability factors.

Our Bayesian Network called AMS-BN (Air Mission System - Bayesian Network) follows the McCall quality model [McCall 1994]. The attributes comprising *quality criteria* are broken into two layers: Level 1 and Level 2. The purpose is to establish a hierarchy of architecture attributes (characteristics), so that more complex attributes can be broken down into simpler measurable attributes or attributes that can be assessed against a checklist. The hierarchy is shown in Figure 2.

The variables representing the quality factors (the user point of view) are given below. In the most simplistic case, we capture them with a range of discrete values {high, medium, low}.

- *Reliability {high, medium, low}* - continuity of correct service;
- *Safety {high, medium, low}* - absence of catastrophic consequences on the users or environment; and
- *Maintainability {high, medium, low}* - ability to undergo modifications and repairs.

Architecture evaluation often accounts only for the architecture characteristics, i.e. the technical aspects. Considering that the maturity of the engineering process and the previous experience of an organisation have significant impact on the outcome of projects in the avionics domain, we define and add attributes characterising the organisation. Thus we arrive at a set of software architecture attributes associated with the ubiquitous triangle – People, Process and Technology (Architecture).

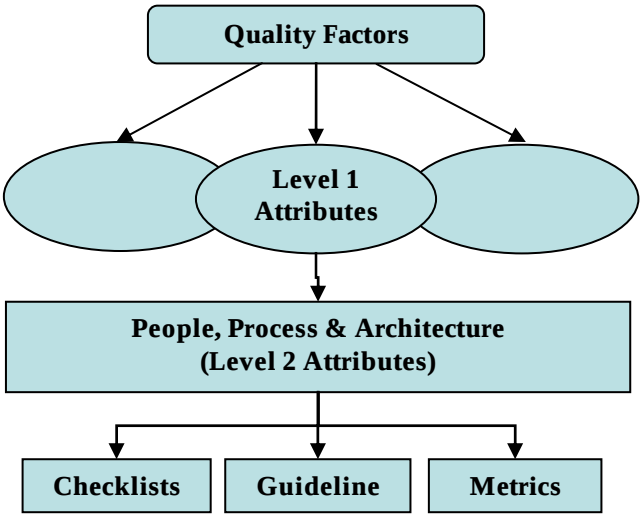

Figure 2: AMS quality measurement framework

3.1.1 Level 1 Criteria (Attributes)

The following list represents the Level 1 quality criteria related to the dependability factors. The causal relationships between the criteria and the chosen attributes are shown in Table 1.

1. *Organisational capability {high, low}* - reflects the level of maturity of the organisation and the project.
2. *Modularity: {high, low}.* This is the degree to which a system or program is composed of discrete components, such that a change to one component has minimal impact on other components [IEEE-100]. A highly modular architecture enables upgrades and selective replacement of modules at a lower cost. It will provide for better error containment and will allow gradual system integration.
3. *Complexity {high, low}.* This criterion reflects whether the architecture is perceived as complex [Gurp 2003]. Increase in complexity will negatively impact all dependability attributes. Correlations between complexity and errors and between complexity and difficulty to understand software are established in [Watson et al. 1996]. Considering that more complex software is more difficult to understand and that exhaustive testing may not be possible (complexity makes software testing harder), a high degree of complexity will ultimately impact on the reliability of a system. As pointed at [Watson et al. 1996], for a fixed level of effort, complexity measures reliability itself.
4. *Modifiability {high, low}* – This criterion reflects the ability of the architecture to accommodate changes – upgrades, substitutes, replacements.
5. *Scalability {high, low}* - The ability to provide functionality up and down a graduated series of application platforms that differ in speed and capacity [IEEE-100], i.e. this criterion reflects the ability of the system / architecture to increase its performance when new hardware or software components are installed.
6. *Openness {yes, no}* - Open architecture is such architecture for which design parameters and specifications are made available to any and all vendors or manufacturing firms, thus encouraging development of compatible products and enhancements.
7. *Fault tolerance {high, medium, low}* – This characterises the ability of a system or component to continue normal operation despite the presence of hardware or software faults. It is based on Error Detection and System Recovery (rollback, roll forward, compensation, diagnosis, isolation, reconfiguration and re-initialisation). The choice of error detection, error and fault handling depends on the fault assumptions. The criterion takes into account the implementation of diagnostic procedures, built-in tests, support for graceful degradation following specified priority and ongoing management of the system health.
8. *Robustness {high, medium, low}* – Robustness is the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [IEEE-100]. We need to view robustness not only as a characteristic of a standalone system, but to anticipate that the modern AMS is designated to act within a system-of-systems.
9. *Testability {high, medium, low}* - The degree to which the design of a system or component facilitates the establishment of test criteria and test execution.

Table 1 represents the mapping of Level 1 quality attributes to quality factors. The table shows only quality attributes that are directly connected to quality factors. It should be noted that some attributes influence different factors in opposite direction; for example implementing a certain degree of fault tolerance improves reliability, while at the same time increases complexity, which in turn negatively impacts testability and reliability.

Table 1: Quality Factors vs. Criteria

	Reliability	**Maintainability**	**Safety**
Modifiability		X	
Modularity	X	X	X
Openness	X		
Organisational capability	X	X	X
Robustness	X		X
Scalability		X	
Testability	X	X	X
Fault tolerance			X

3.1.2 People Related Level 2 Attributes

Most of the technologies and the theories of software engineering are human based, and, as such, depend on the variations of skill level, competence and motivation. The provenance of open source software is one example of people related aspects that may be considered during evaluation. In addition to the developer / manufacturer issues, there are human factors and concerns on the user / operator side, where the operator can be viewed as an integral component of the system or as an entity outside of the system. Of all possible criteria we have selected only three which are expected to capture the competency of the organisation and developers.

1. *Experience within the domain* - reflects the level of experience the development organisation has with the platform and / or with a given application. {Low (experience less than 2 years), Nominal (2-5 years), and High (more than 5 years)}.
2. *Skill retention* - reflects how well the skills are retained on the project. We use the annual turnover rate to characterise this variable: {High retention (turnover rate of 5-6% or less), Low (turnover rate of 15-20% and more)}.
3. *Management practices* – project management approaches can increase the technical risk for a project [Goldsmith et al. 2008]. Risk tolerance, skill and experience of the management team will impact the outcome of a project. We apply two ratings: {Low (poor) and High (good)}.

3.1.3 Process Related Level 2 Attributes

1. Process maturity

The impact a process maturity has on the dependability can be illustrated by the phase containment effectiveness and customer reported defects as shown in Table 2 [Diaz et al. 2002].

We will use the Capability Maturity Model Integration (CMMI) maturity level as a quantitative measure of the quality of the process within a project. We can use the results from appraisal and self assessment as evidence of the maturity level, or we can "assign" a level based on analysis of the available project plans, development documentation, project reviews and minutes. The process maturity variable may have one of the following states: {Level_3_5 and Level_1_2}.

Other equivalent process or quality standard would be TickIt, SPICE, and ISO 9002. They define documentation and actions required to deliver quality software.

Table 2: General Dynamics Decision Systems Project Performance versus CMM level

CMM Level	Percent Rework	Phase Containment Effectiveness	Customer Reported Unique Defects Density per KLOC	Productivity (X Factor Relative)
2	23.2%	25.5%	3.20	1x
3	14.3%	41.5%	0.90	2x
4	9.5%	62.3%	0.22	1.9x
5	6.8%	87.3%	0.19	2.9x

2. *Relevant standards*

Standards such as RTCA DO-178 (RTCA DO-278), the Software Development Standard for Space Systems (SDSSS), IEEE 982.2 "IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software", Def(Aust) 5679 etc. impact dependability, especially if used within a mature organisation. The application of a relevant standard would decrease the number of defects in code and documentation and thus improve reliability, availability and safety. In order to come up with a concrete number we used a CoComo based tool (CostXpert) to provide a crude guideline. The tool is providing an estimation of the number of defects for selected type of projects (embedded, military etc), lifecycle and applied standards. We realise that selecting the type of project may define quite a loose framework, but experimenting with different combination we assume that the application of an appropriate standard would translate into a 12-15% decrease of defects (in code or documentation). The variable has two states {yes, no}.

3.1.4 Technology / Product Related Level 2 Attributes

1. *Architecture style/patterns*

The architectural patterns are one of the techniques for designing high quality software architectures. As pointed in [Booch 2006] we are beginning to observe the emergence of domain-specific software architectures. They represent reusable solutions which evolved within different application domains. Different patterns will have a different impact on the reliability, maintainability and performance of the architecture. Based on practical experience, [Buschman et al. 1996] grouped the architecture structures in several classes (patterns), namely, Layered, Pipes and Filters, Blackboard, Distributed Systems Broker, Model-View-Controller, Presentation-Abstraction-Control and Microkernel. We will refer the reader to the book for further details on these patterns, the context they are most suitable for, and how they relate to

the software process and various software development techniques. We adopted this classification scheme because it is practical and can cover most of the existing architectures under investigation.

Svanberg and Wohlin interviewed a group of software architects and captured the data for a consequent analysis and comparison between software architectures with respect to selected quality attributes: efficiency, functionality, usability, reliability, maintainability and portability [Svahnberg et al. 2005]. Although the main concern of this work was the development of a methodology for eliciting the views of different stakeholders, thereby formulating a framework for quality attributes and analysis of different architectural structures, we consider the data captured by the interviews as solid quantitative expert assessment which can be used to fill in the Node Probability Table for the variable "Architecture style". According to the study, the ranking of architecture structures per quality attribute is presented in Table 3 (we quote only the attributes of interest to us). One suggestion offered by the report is that no architecture is really strong in, or, is focused on, some specific quality attributes (reliability being one of them). Consequently, we give relatively low weight to the variable "Architecture style".

Table 3: Framework for quality attributes as in [Svahnberg et al. 2005]

Quality Attribute	Microkernel	Blackboard	Layered	Model-View-Controller	Pipes and Filters
Functionality	0.228	0.261	0.179	0.188	0.144
Reliability	0.207	0.103	0.270	0.239	0.180
Maintainability	0.124	0.171	0.283	0.198	0.225
Portability	0.194	0.0767	0.366	0.157	0.207

2. *Quality of requirements elicitation: {high, low}.* Some of the causes for project failure stem from this phase of the development cycle, i.e. *Ad hoc* requirements management, ambiguous and imprecise communication and undetected inconsistencies in requirements, designs, and implementations.
3. *Cohesion {strong, weak}* – The manner and degree to which the tasks performed by a single software module are related to one another [IEEE-100].
4. *Coupling {tight, loose}* – The manner and degree of interdependence between software modules. Aspects to be considered may be dependencies on common environment, content coupling, control coupling, data coupling etc.
5. *Change propagation/containment {yes, no}* – functionality is mapped to components / modules in such a way that a change / upgrade in functionality will affect only known and isolated components and not affect others implementing different functionality.
6. *Error propagation/containment {high, low}* – Error propagation from one component / module to another reflects the likelihood that an error in the first component will cause an error in the second component (considering that the first component feeds information to the second). Low value for this attribute means that a

failure in any component / module will not result in unintended failures in other system modules.

7. *Views documentation {documented, not-documented}* – reflects the quality of documenting the architecture. The various high-level aspects of software architecture are often called views: "A view represents a partial aspect of a software architecture that shows specific properties of a software system" [Buschman et al. 1996]. [SWEBOK 2004] lists different sets of views that have been suggested; the evaluation of this attribute, however, should be concerned with how complete, clear and comprehensive is the documentation and if it captures well the requirements and elaborates them for the next stage of the software development cycle.
8. *Capacity {high, nominal, low}* – The ability to add new functionality or extend the existing one will depend on the resource usage such as CPU (peak CPU load, spare capacity), power consumption, I-O rates (e.g. expected network usage), the processor load, memory / HDD utilisation, network loads.
9. *Interoperability {yes, no}* – This criterion defines the degree to which information or services can be exchanged to enable them to operate effectively together. The requirements for interoperability will impact the scalability, complexity and ultimately the reliability and maintainability of the software architecture.
10. *HW-isolation {yes, no}* – reflects the degree to which the application layer is isolated from the hardware specifics. This attribute covers questions such as: (1) is there a hardware abstraction layer; (2) do applications use low-level constructs (e.g. no assembler) or access hardware directly; (3) does the software rely on graphics accelerators that may inhibit portability etc.
11. *Scheduling {deterministic, non-deterministic}* – the scheduling algorithm in embedded systems will affect future upgrades. For example, rate-monotonic scheduling gives deterministic guarantees with regards to response times.
12. *Interfaces {open, proprietary}* – This attribute reflects whether well defined, widely used, non-proprietary interfaces and protocols are used. In relation to networks, it reflects whether open network standards and COTS components are used.
13. *Technology independence {yes, no}* – This attribute reflects the extent to which the architecture depends on current technology, e.g. application of standards and established COTS products, availability of DMSMS plans and technolog1y refresh plans, identification of coding standards etc.
14. *Fault prevention {yes, no}* – aims to avoid fault occurrences by construction, i.e. information hiding, usage of strongly-typed programming languages etc.
15. *Memory management {good, bad}* – the attribute reflects whether there is dynamic or static allocation of memory.
16. *Task management {controlled, dynamic}* – reflects specifics of task management such as how the application manages tasks, creation of dynamic tasks, deletion of tasks, assignment of task priorities.

3.2 Dependencies Definition

Each node in a Bayesian Network is associated with a CPT which maps the probabilities of a node to each configuration of parent values.

For root nodes the CPT represents basic knowledge (or opinion, beliefs). The CPT table for root nodes is assigned values based on knowledge about a specific architecture which is gained as a result of architecture reviews, audits, analysis of available documentation etc. To help with this process a checklist is associated with every root node. The checklist is expected to help the user and is at the same time a tool to capture the knowledge of experts in the relevant field. We will not go through all nodes, but just to demonstrate the logic we follow while filling in CPT, we will just give an example of the CPT and checklist associated with one of the root nodes (attribute) and one intermediate node (criteria).

The checklist shown in Table 4 is based on [Goldsmith et al. 2008] and lists a number of concerns when analysing the node 'Management Practices'. It is expected that the user of the model will make an assessment using such checklists in addition to other relevant information before entering values in the relevant CPT.

Table 4: Management practices attribute – checklist of concerns to be addressed

Quality Criteria	Checklists
Management practices	1. Is there increasing contingency (funding and schedule contingency increase acceptance of risk) 2. Compromising engineering rigour in order to meet schedule / cost 3. Delegation / assignment risk to a contractor or another project 4. Is there deferred effort (deferred effort tends to realise technical risks). Typical violations are writing design documents post coding effort, retrofitting certification evidence after the system is developed etc. 5. Is the risk mitigation activity resourced properly 6. Is the estimation based on realistic assumptions or is it a best case estimate (e.g. assuming the participation of "A" team) 7. Is key system analysis completed before starting

Two more examples are the CPT for 'Skill retention' and 'Process maturity'.

Skill retention: According to [Roman 2007] the average annual turnover rate for the semiconductor industry is 7% (being 7.5% in the past few years) and ranging from high 12% to less than 1%. [AEA 1995] shows that the turnover rate for all engineers in the Electronics and Information Technology companies was 11.4% compared with 9.5% in 1993 and 9.7% in 1992. For 1998 AEA reported an average turnover rate of 21.8% for electronics engineers at its member companies, but 25.5% for software and programmer analysts and 23.7% for software engineers. Based on these figures, we defined our expectation for the attribute 'Skill retention'.

Process maturity: According to [CMMI_SEI 2006], out of 1377 reporting organisations 18.2% are at Level 5, 4.4% are at level 4, 33.8% are at level 3, 33.3% are at level 2, 1.9% are at level 1 and 8.4% did not respond. We can use this data to initially define the CPT – 38.24% probability for Level_1_2 and 61.76% probability for Level_3_5. Considering that the reported profile is not drastically different from a similar one produced in 2004, the assumption should be realistic.

When we have a specific project in mind, then we would be able to assign 100% probability to one of the values of 'process maturity' and 0% to the other value.

CPT for the node 'Organisation capability' is shown in Table 5. The numbers reflect our opinion which is derived from experience and relevant publications. According to [Jones 2000], the experience levels of both the managers and the technical staff in building similar type of applications has a combined impact on productivity of 120%, while effective methods / processes have a positive impact of only 35%. The following reverse effect is also cited: among the factors that can reduce or degrade software productivity, management and staff inexperience have a combined negative impact of 177%, while ineffective methods / processes negative impact is 41%. Based on this, we will attribute more weight to the 'Skill retention' and 'Management practices', than 'Process maturity'.

Table 5: CPT for the node 'Organisation capability'

Process	Skill/ retention	Management practices	Architecture quality	Organisation capability H	Organisation capability L
Level_3_5	Nominal	High	High	90%	10%
Level_3_5	Nominal	High	Low	70%	30%
Level_3_5	Nominal	Low	High	75%	25%
Level_3_5	Nominal	Low	Low	30%	70%
Level_3_5	Low	High	High	85%	15%
Level_3_5	Low	High	Low	65%	35%
Level_3_5	Low	Low	High	70%	30%
Level_3_5	Low	Low	Low	25%	75%
Level_1_2	Nominal	High	High	65%	35%
Level_1_2	Nominal	High	Low	35%	65%
Level_1_2	Nominal	Low	High	40%	60%
Level_1_2	Nominal	Low	Low	15%	85%
Level_1_2	Low	High	High	60%	40%
Level_1_2	Low	High	Low	30%	70%
Level_1_2	Low	Low	High	35%	65%
Level_1_2	Low	Low	Low	10%	90%

3.3 Application of the Airborne Mission Systems BN Model

The probabilities generated by the model are expected to be interpreted as a guideline, i.e. we are looking for the trend whether specific probability is high or low. Based on this, a conclusion can be made about the associated risk. The result of the architecture evaluation using the AMS_BN model will depend on the quality of the assessment of the root nodes. The expert needs to consider the existing evidence and define the CPT for the root nodes. In the absence of enough evidence about a specific root node, a decision needs to be made whether to (1) use the default CPT (which aims to represent industry averages), or (2) define CPT with more conservative numbers, or (3) re-consider the impact of this root node on its children.

4. Conclusions and Future Work

At the phase of architecture definition a direct evaluation of dependability is not possible, however, some information is already available, e.g. the quality of the software development process, software standards applied, project environment, analysis of requirement specifications, architecture level models, expert opinion etc. This evidence is uncertain and incomplete: none of it, by itself, can provide enough information about the quality of software. Therefore, in order to build a holistic picture of the state of dependability, a type of causal model – Bayesian Network model, is proposed. The AMS-BN model considers evidence generated up to (and including) the architecture definition phase. The topology and the dependencies definitions of the AMS-BN are described. This work demonstrates the application of BN approach in the Airborne Mission Systems domain. The created model is aimed to assist the AMS team with the technical risk assessment of mission system software in Defence acquisition projects. Netica [Netica RM 2007] is used to run the model.

The AMS-BN model is currently being applied to a Defence project, and, when the project is completed, the model prediction will be compared with the project outcomes. This will be used as a validation approach, in addition to peer reviews and checking that the results match common sense and observed results from previous projects.

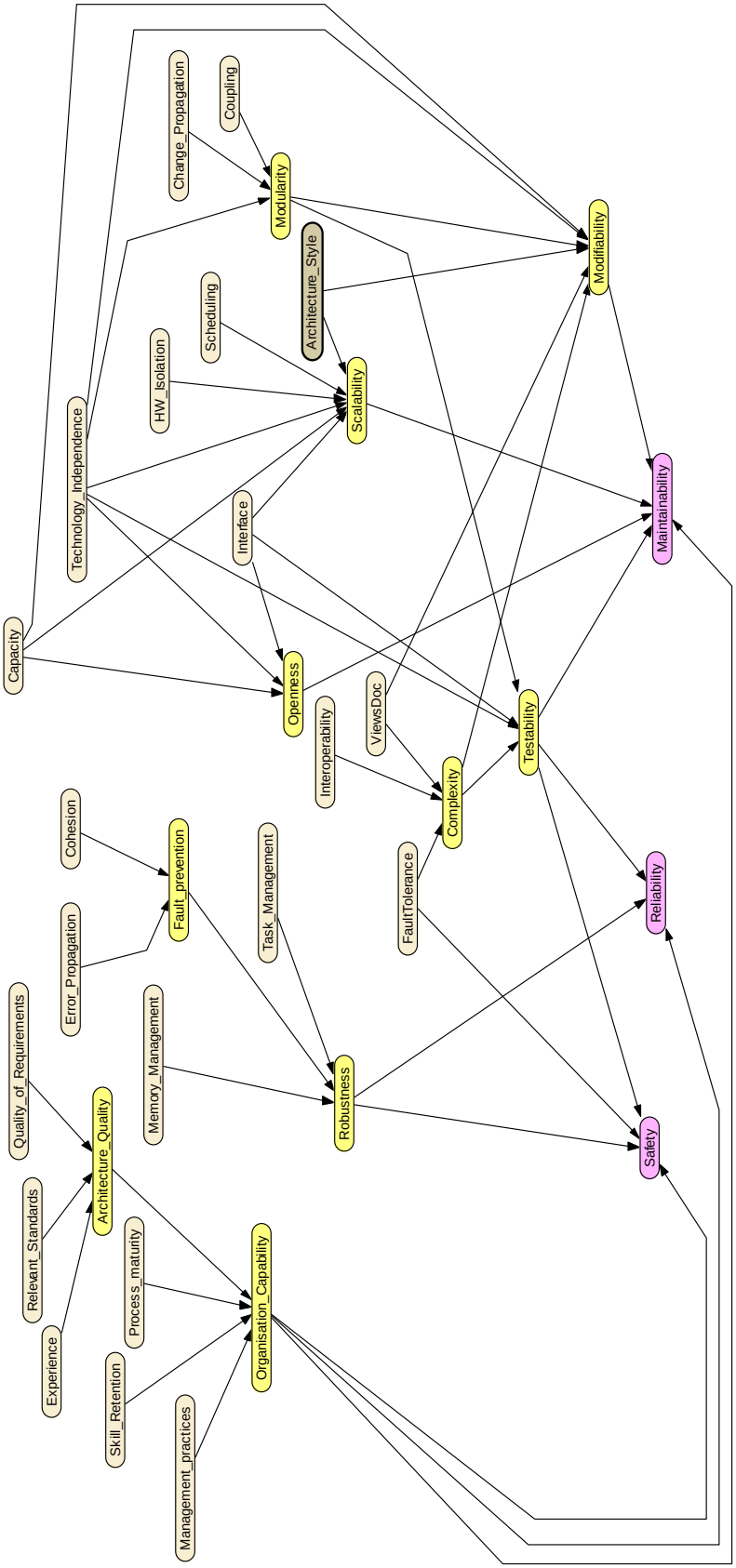

Figure 3: AMS-BN connectivity

5. References

AEA 1995 Engineering Compensation Survey Shows Jump in Turnover Rate, Business Wire, 15 November 1995

Anthony, KD. Introduction to Causal Modeling, Bayesian Theory and Major Bayesian Modeling Tools for the Intelligence Analyst, USAF National Air and Space Intelligence Center (NASIC), November 2006, v.0.94 http://www.au.af.mil/au/awc/awcgate/nasic/intro-causal-modeling.pdf

Avizienis, A, Laprie J, Randell, B and Landwehr, C. "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Trans. On Dependable and Secure Computing, Vol.1, No1, pp. 11-33, Jan-Mar 2004

Babar, M, Zhu, L and Jeffrey, R. A Framework for Classifying and Comparing Software Architecture Evaluation Methods, Proc. Of the 2004 Australian Software Engineering Conference (ASWEC'04), 10p., 2004

Bergner, K, Rausch, A, Sihiling M and Ternite, T. DoSAM – Domain-Specific Software Architecture Comparison Model, in Reussner R, et al. (Eds.):QoSA-SOQUA 2005, Springer-Verlag LNCS 3712, pp. 4-20, 2005

Bass, L, Clements, P and Kazman, R. Software Architecture in Practice, 2nd ed., Addison-Wesley, 2003

Booch, G. The Future of Software, Systems and Software Technology Conference (SSTC), 2006, A keynote speech

Bosch, J. Presentation on Software Architecture Assessment, Summer School on Software Architecture, Turku Centre for Computer Science, Finland, August 2001

Bosch J and Bengtsson, P. Assessing Optimal Software Architecture Maintainability, Fifth European Conference on Software Maintenance and Reengineering, pp. 168-176, 2001

Buschman, F, Meunier, R, Rohnert, H, Sommerland, P and Stal, M. Pattern-Oriented Software Architecture, John Wiley & Sons, Chichester UK, 1996

Charniak, E. Bayesian Networks without Tears, A publication of the AAAI (American Association of Artificial Intelligence), 63 p., 1991

CMMI_SEI 2006, Software Engineering Institute, Capability Maturity Model Integration (CMMI) Version 1.2 Overview, CMU, 41 p., 2006

Diaz, M and King, J. How CMM Impacts Quality, Productivity, Rework, and the Bottom Line, Crosstalk: The Journal of Defense Software Engineering, pp. 9-14, March 2002

DoD DAG, US DoD, Defence Acquisition Guidebook, https://akss.dau.mil/dag/

Falla, M. ed., Advances in Safety Critical Systems: Results and Achievements from the DTI/EPSRC R&D Programme in Safety Critical Systems, 1998. http://www.comp.lancs.ac.uk/computing/resources/scs/

Fenton, NE, Littlewood, B, Neil, M, Sutcliffe, A and Wright, D. Assessing Dependability of Safety Critical Systems Using Diverse Evidence, IEEE Proceedings on Software Engineering, vol.145, No.1, pp. 35-39, February 1998

Fenton, NE and Neil, M. Managing Risk in the Modern World: Bayesian Networks and the Applications, London Mathematical Society, Knowledge Transfer Report, November 2007

Firesmith, D. QUality Assessment of System ARchitectures (QUASAR), SEI CMU presentation, ver. 0.1, 2006. http://www.sei.cmu.edu/programs/acquisition-support/presentations/quasar.pdf

Florentz, B and Huhn, M. Embedded Systems Architecture: Evaluation and Analysis, in C.Hofmeister et al. (Eds.): QoSA 2006, Springer-Verlag LNCS 4214, pp. 145-162, 2006

Goldsmith, K O'Dowd, R and Davis, M. Technical Risk Assessment, DSTO AMS Seminar Series, 49 p., 2008

Gregoriades, A and Sutcliffe, A. Scenario Based Assessment of Non-functional Requirements, IEEE Transactions on Software Engineering, 31(5), pp. 392-409, May 2005,

Gurp, J. SAABNet: Managing Qualitative Knowledge in Software Architecture Assessment, from "On the Design & Preservation of Software Systems", PhD Thesis, pp. 73-88, 2003

IEC 9126, AS/NZS ISO/IEC 9126 Standard, Software Engineering – Product Quality, 2005

IEEE-100, IEEE, IEEE-100 the Authoritative Dictionary of IEEE Standards Terms, Seventh Edition, IEEE Press, ISBN-0-7381-2601-2, 2000

IEEE/EIA 12207, IEEE/EIA, Standard for Information Technology-Software Life Cycle Processes, IEEE, 2008

Jones, C. Software Assessments, Benchmarks, and Best Practices, Addison-Wesley, 659 pages, 2000

Korb, KB and Nicholson, AE. Bayesian Artificial Intelligence, Chapman & Hall/CRC, 364 p., 2004

Lundberg, L, Bosch, J, Haggander, D and Bengtsson, P-O. Quality Attributes in Software Architecture Design, Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications, Arizona, USA, pp. 353-362, October 1999

McCall, JA. Quality Factors, In „Encyclopedia of Software Engineering", vol.2 O-Z, Marciniak, JJ. ed., John Wiley & Sons Inc., pp. 958-969, 1994

MoD 2006, Ministry of Defence, Defence Technology Strategy for the demands of the 21 century, 2006

Murphy, K. Software Packages for Graphical Models/Bayesian Networks, Last updated 31 October 2005 http://www.cs.ubc.ca/~murphyk/Bayes/bnsoft.html

Netica RM 2007, Netica Reference Manual, Norsys Software Corp, v. 3.25, Oct 2004

Perez-Minana E and Gras, J-J. Improving Fault Prediction Using Bayesian networks for the Development of Embedded Software Applications; Software testing, verification and reliability, 16, 2006, pp. 157-174

Prasad, DK. Dependable Systems Integration using Measurement Theory and Decision Analysis, PhD thesis, University of York, 263 pages, November 1998

Roman, D. How to keep engineers happy, 3 pages, EETimes, 18 Jun 2007, http://www.eetimes.com/news/latest/showArticle.jhtml?articleID=199905039

Sandborn, P. Trapped on technology's trailing edge, IEEE Spectrum, pp. 39-43, April 2008

Shereshevsky, M, Ammari, H, Gradetsky, N, Mili, A and Ammar, H. Information Theoretic Metrics for Software Architectures, 2001

Simon, C and Weber, P. Bayesian Networks Implementation of the Dempster Shafer Theory to Model Reliability Uncertainty, Proceeding of the First IEEE International Conference on Availability, Reliability and Security (ARES), 2006, 6 pages

Stineburg, J, Zage, W and Zage, D. Measuring the Effect of Design Decisions on Software Reliability, Software Engineering Research Center (SERC), SERC-TR-272, 19 p., May 2005

Sutcliffe, A and Gregoriades, A. Validating Functional System Requirements with Scenarios, Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02), 2002, September 2002, 8 pages

Svahnberg, M and Wohlin, C. An Investigation of a Method for Identifying a Software Architecture Candidate with Respect to Quality Attributes, Empirical Software Engineering: An International Journal, Vol.10, No. 2, pp. 149-181, 2005

SWEBOK 2004, IEEE Computer Society, Guide to the Software Engineering Body of Knowledge, 2004

Uzunov, K and Nguyen, T. Dependability of Software in Airborne Mission Systems, DSTO Technical Report, DSTO-TR-2111, 59 p., 2008

Voas, J. Software Quality Unpeeled, CrossTalk, The Journal of Defence Software Engineering, pp. 27-30, June 2008

Wang, H, Peng, F, Zhang C and Pietschker, A. Software Project Level Estimation Model Framework based on Bayesian Belief Networks, Sixth International Conference on Quality Software (QSIC'06), pp.209-218

Watson, AH and McCabe, TJ. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, NIST Special Publication 500-235, 123 p., Sep 1996

Ziv, H, Richardson, DJ and Klosch, R. The Uncertainty Principle in Software Engineering, 19th International Conference on Software Engineering ICSE'97, 12 p., 1997

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING / CAVEAT (OF DOCUMENT)

2. TITLE

Evaluation of Software Dependability at the Architecture Definition Stage

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Kiril Uzunov and Thong Nguyen

5. CORPORATE AUTHOR

DSTO Defence Science and Technology Organisation
506 Lorimer St
Fishermans Bend Victoria 3207 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-2428	AR-014-792	Technical Report	June 2010

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
2009 / 1157821	07 / 245	CAOD	21	47

13. URL on the World Wide Web

http://www.dsto.defence.gov.au/corporate/reports/DSTO-TR-2428.pdf

14. RELEASE AUTHORITY

Chief, Air Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DSTO RESEARCH LIBRARY THESAURUS http://web-vic.dsto.defence.gov.au/workareas/library/resources/dsto_thesaurus.shtml

software engineering, software architecture, risk management, Bayesian networks

19. ABSTRACT

The problem we aim to solve is how to evaluate the dependability of software at the stage of architecture definition. Evidence, such as the process maturity, project environment and architecture documentation is already available and can be used for the evaluation. In order to create a holistic picture of the state of dependability, a Bayesian Network (BN) model is defined. The paper defines a quality framework which guides the model creation, identifies attributes characterising dependability and presents the topology of the model. The approach to the quantitative definition of the model is illustarted by examples. The model is aimed to help with conducting technical risk assessment of Airborne Mission Systems.